Line Integral Convolution

Chris Parker. Universitetet i Oslo. UNIK 4660. Spring 2015.

Source code and images available at:
github.com/csp256/LineIntegralConvolution

- Introduction

I implemented forward Euler and the standard Runge-Kutta (RK) algorithm for computing streamlines on 2D velocity field data stored in a HD5 file. I also implemented visualization of discrete streamlines and line integral convolution (LIC). These are two qualitatively different approaches towards visualization of flow: discrete sampling and texture based. Both methods performed well and created clear graphs, optionally including additional data such as flow magnitude. It is important to note here that “performed well” is done within the context that visualization metrics are inherently subjective.

- General Discussion

While the simulator data is more “clean”, the Isabel dataset was used for testing as it was considered more representative of real data, and contained an edge case not found in the simulator dataset: points of zero velocity. This was extracted into a mask and the relevant pixels were ignored. In all cases all random variables (such as that for the input noise texture were) seeded for reproducible results. In all cases vector magnitude was ignored in computing the streamline. Linear and cubic interpolation of the input vector field was implemented, but no significant gains were observed from the slower cubic interpolation.

The noise texture was never interpolated. It was found that discretizing the streamline's convolution at a resolution width higher than the pixel width significantly decreased results, regardless of integration method. It was also found that while the increased accuracy of the RK algorithm is extremely desirable in simulation due to the fifth order residual, decoupling the convolution spacing from the integration points (to permit a larger step size) using cubic interpolation produced visually similar results, and did not permit a lower minimum number of steps. In fact, for reasonable convolution widths (typically in [5, 15]) there was no meaningful difference between forward Euler and the Runge Kutta method, while the first was faster/simpler. This is because relevant errors are only propagated over the convolution width and texture sampling was done on a basis (equivalent to) nearest-pixel. That the velocity field data is interpolated with a lower accuracy than the integration method provides further justification for the counter-intuitive superiority-through-simplicity of the (typically inferior) forward Euler… to say nothing of the subjectivity of texturing methods.

- Discrete streamline implementation

A low discrepancy sequence (Sobolev sequence), also called simulation noise for its attractive properties, was used to seed streamlines. This allowed points to be chosen semi-randomly while being of a uniform density. An outside implementation of this algorithm was used, and is clearly disclaimed in the source. Its inclusion resulted in a better (unbiased) representation of the streamlines. These streamlines were then superimposed over the magnitude using simple alpha blending.

However, in cases of circular flow (zero divergence) streamlines do not show flow direction at all, and

only poorly in low divergence fields. This motivates the LIC algorithm. Additionally, even with semi-uniform sampling some areas are under represented by the discrete streamline method, as in the corners of the image “metsim_low_discrepency”.

- LIC implementation

In the LIC algorithm a noise texture of identical resolution to the velocity field was created (though in theory different resolutions are possible or even preferable). A sequence of points are selected. For each point, a bounded length streamline is computed. It extends “L+M” many integration points of width “h” (experiments found values of 1 or slightly smaller worked best), where L is the convolution width (“radius”) and M is the padding in each direction (equivalently, it is proportional to the number of textures per streamline). In general, higher values of M and lower values of L increased computational efficiency, though L could not be decreased without bound and arbitrarily high values of M became wasteful as potentially dozens of streamlines contributed to the same texture pixel. Broadly, efficiency went as M / (L+M).

Initially these seed points for streamline integration were chosen by a low discrepancy sequence, and then brute force iteration over the entire image until the number of “hits” (streamlines contributing to a texture pixel) was higher than some bound. A value of 3 was found to be more than sufficient for visual fidelity. Seeding by low-discrepancy sequence increased the effectiveness of pixel correlation along regions of low stream density (there was a clearer directionality with fewer minimum stream numbers if they were seeded randomly first, especially with high streamline length), and prevented aliasing from sequential processing becoming apparent.

The number of streamlines per texture pixel was stored in a separate field, and used to normalize the sum of texture pixels. Each term in the integral of each texture pixel was done in a fast manner. In practice this meant that fast convolution was done by “pushing and popping” values from the front and back of the convolution window along the streamlime in both directions. This was permissible & fast because the noise image was convolved against a constant-valued function. Perhaps this variant of the LIC algorithm is thus better termed a “streamline segment integral texture”, as the integral was of this form[1]:

$$I(\boldsymbol{x}_2) = I(\boldsymbol{x}_1) \; - \; k\int_{s_1-L}^{s_1-L+\Delta s} T(\boldsymbol{\sigma}(s))\,ds \; + \; k\int_{s_1+L}^{s_1+L+\Delta s} T(\boldsymbol{\sigma}(s))\,ds.$$

On each brute-force iteration all seed points had both axes perturbed by unit i.i.d. noise that was shared amongst all points: this shifted the entire lattice of seed points by a constant amount and helped to prevent redundant computations on additional full image passes. On passes after the first M was decreased to decrease redundancy of streamlines extending to well covered areas.

That is, up to minor details, the algorithm specification in [1] was used:

for each pixel p
    if (numHits(p) < minNumHits) then
        initiate stream line computation with $\boldsymbol{x}_0$ = center of p
        compute convolution $I(\boldsymbol{x}_0)$
        add result to pixel p
        set $m = 1$
        while $m <$ some limit M
            update convolution to obtain $I(\boldsymbol{x}_m)$ and $I(\boldsymbol{x}_{-m})$
            add results to pixels containing $\boldsymbol{x}_m$ and $\boldsymbol{x}_{-m}$
            set $m = m + 1$
for each pixel p
    normalize intensity according to numHits(p)

- Observations

The LIC algorithm tended to create a texture with values almost all within [0.25, 0.75]. I found that by rescaling this field so that its maximum is 1 (to make it independent of the distribution width, which is a function of L), and then biasing it by a number (usually within) [0, 3] that I could smooth the texture and make it contribute more subtly to the colored field. The images suffixed with “shifted” and “unshifted” highlight this. The image suffixed with “shifted” is one of the nicest I produced, because it clearly shows flow direction while minimally perturbing the apparent magnitude.

Specifically, the spectral and Paired colormaps provided high chromatic discrepancy with periodic luminosity that is well-suited for general purpose qualitative appraisal of textured vector fields.

The most clear visualizations of streamline direction was achieved by neglecting the magnitude information altogether and solely visualizing the texture field. Unfortunately, this looses critical features: it is not clear that the eye of the hurricane is nearly still. This is because of the re-normalization of vector field intensity in the streamline integration, and a primary reason why I chose to work with the (larger, slower) Isabel dataset.

Unfortunately, it can be difficult to tell if an artifact along a streamline is a result of the texture field (random chance) or is an actual feature of the velocity field. Recomputing the image with a new texture seed and comparing (or animating) is the simplest solution to this problem (see the images with “seed_0” and “seed_1” in their file name) but is not preferable due to being computationally wasteful. Many more sophisticated (complicated) methods for animating the streamlines exist, but were not implemented (though this is the most clear extension of the work, beyond porting it to a higher performance language).

- Language justification

I used an iPython notebook because I was interested in the trade offs between slow execution speeds and fast development times on a computationally intensive task. Also, the interactivity of the Jupyter & numpy frameworks is not something I had experimented with previous, and is a standard of my field. As a person who “writes C code in every language”, who has extensive experience with GPGPU programming, and always optimizes prematurely, I found that I was more productive than expected in Python, even without much prior experience in the language and the slow run times. In particular I found I could hide almost all execution time by developing the two algorithms in parallel: while the results from one algorithm was being computed I would edit the other algorithm using the insight gained from the previous computation's results.

- Conclusion

Each algorithm has its applications. The discrete sampling method was very easy to implement and quite fast. However, it did a poor job in regions of low divergence and could not fully resolve, but instead only hint at, smaller scale features.

The LIC algorithm is computationally expensive in principle (regardless of my implementation), though suitable for real time interaction with modern computers. Displaying just the texture provided the highest information density about flow direction, but could be misleading by obscuring flow magnitude discrepencies. Element-wise multiplication of the texture image and the magnitude image partially remedied this. However, it distorted the magnitude image and created strong artifacts along the streamlines. Biasing the texture image prior to combining it with the magnitude image permitted more subtle texturing to be achieved, while better preserving the fidelity of the magnitude image.

Surprisingly, Runge Kutta integration did not prove to be a significant qualitative improvement over forward Euler, despite its quantitative superiority. Perhaps second order Runge Kutta methods could provide a better compromise for accuracy, even though interpolating between integration points was not found to be worthwhile.

[1] D. Stalling, and H. Hege, “Fast and resolution independent line integral convolution.” *1995.*
[2] B. Cabral, and L Leedom. “Imaging vector fields using line integral convolution.” *1993.*